CSCI 31072 – Python Programming

Assignment 01

Pawan Pinsara Perera

CS/2020/005

GitHub Link for the Code:

https://github.com/PinsaraPerera/Python_Assignment.git

Exercise 01: Library Management System

Code:

```
class Book:
    def __init__(self, title, author, isbn=None): # initialize the book with title, author, and isbn number
        self.title = title
        self.author = author
        self.isbn = isbn

class Library:
    def __init__(self):  # initialize the library with an empty list of books
        self.books = []

    def add_book(self, book):
        self.books.append(book) # add book to the library

        with open('books.txt', 'a') as file:
            file.write(f'{book.title}, {book.author}, {book.isbn}\n') # store each book in books.txt while adding it to the library

    def find_books(self, title=None, author=None): # find books by title or author
        found_books = []
        for book in self.books:
            if (title and book.title == title) or (author and book.author == author):
                found_books.append(book)
        return found_books

    def remove_book(self, title=None, author=None): # remove books by title or author
        for book in self.books:
            if (title and book.title == title) or (author and book.author == author):
                self.books.remove(book)
                return book

book1 = Book("The Load Of The Rings", "J.R.R. Tolkien", "9780544003415")
book2 = Book("Rich Dad Poor Dad", "Robert T. Kiyosaki", "9781612680194")
library = Library()

# add book to the library
```

```
library.add_book(book1)
library.add_book(book2)

# find the book by title and author
book_find = library.find_books(title="The Load Of The Rings")
print(book_find[0].title)

book_find = library.find_books(author="Robert T. Kiyosaki")
print(book_find[0].title)

# remove the book by title
book = library.remove_book(title="The Load Of The Rings")
print(book.title + " has been removed")

book = library.remove_book(author="Robert T. Kiyosaki")
print(book.title + " has been removed")
```

Explanation:

First, I create Book objects names book1 and book2 for the books that I want to store. Then I create a Library object name library to initialize the library. Then I store my book objects by calling add_book method in library object. I set up to store books' detail inside of books.txt while its adding to the library. Then I return the book lists by using find_book method by filtering the books by its title and the author. Finally, I showed how to remove books from the library using remove_book method.

Output:

$ python exercise_01.py

The Load Of The Rings

Rich Dad Poor Dad

The Load Of The Rings has been removed

Rich Dad Poor Dad has been removed

books.txt:

The Load Of The Rings, J.R.R. Tolkien, 9780544003415

Rich Dad Poor Dad, Robert T. Kiyosaki, 9781612680194

Exercise 02: Student Grade System

Code :

```python
import json

class Student:
    def __init__(self, id, name, grades = []):
        self.name = name
        self.id = id
        self.grades = grades
        self.average = 0

    def add_grade(self, grade):
        self.grades.append(grade)

    def get_average(self):
        self.average = sum(self.grades) / len(self.grades)
        return self.average

    def status(self):
        if self.average >= 60:
            return "pass"
        else:
            return "fail"

    def store(self):

        student = {
            "id": self.id,
            "name": self.name,
            "grades": self.grades,
            "average": self.average,
            "status": self.status()
        }

        with open("students.json", "w") as file:
            file.write(f"{json.dumps(student)}\n")

# initialize the student object
student = Student(1, "Pawan", [90, 85, 84, 75, 60])
```

```python
# add grade to the student
student.add_grade(80)

# get the average of the student
average = student.get_average()
print(f"Average of student {student.name} is: {average}")

# get the status of the student
status = student.status()
print(f"Status of student {student.name} is: {status}")

# store the student data in JSON format
student.store()
```

Explanation:

First, I create a Student object names student and then add grade 80 by calling add_grade method. Then by calling get_average method calculate the students' average mark. Then base on average mark using conditional if clause decided whether student pass or fail, and store the status in self.status variable. Finally, by calling store method store the student details in students.json file. I especially need to mention that even though we can store the student dictionary as a json file I use external Json library to convert the student dictionary object to Json object by json.dump method.

Output:

$ python exercise_02.py

Average of student Pawan is: 79.0

Status of student Pawan is: pass

students.json:

```json
{
    "id": 1,
    "name": "Pawan",
    "grades": [90, 85, 84, 75, 60, 80],
    "average": 79.0,
    "status": "pass"
}
```

Exercise 03: Task Manager

Code:

```python
class Task:
    def __init__(self, title, description, status):
        self.title = title
        self.description = description
        self.status = status

class TaskManager:
    def __init__(self):
        self.queue = []

    def add_task(self, title, description, status):
        task = Task(title, description, status)
        self.queue.append(task)

    def get_task_by_status(self, status = "Pending"):
        tasks = []
        for task in self.queue:
            if task.status == status:
                tasks.append(task)
        return tasks

    def get_task_by_title(self, title):
        for task in self.queue:
            if task.title == title:
                return task
        return None

    def change_task_status(self, title, status = "Completed"):
        task = self.get_task_by_title(title)
        if task:
            task.status = status
        else:
            print(f"Task with title {title} not found")

    def store_task_in_csv(self):
        with open("tasks.csv", "a") as file:
            for task in self.queue:
                file.write(f"{task.title},{task.description},{task.status}\n")
```

```python
# Initialize the Task Manager by creating a new object manager
manager = TaskManager()

# Add tasks to the Task queue
manager.add_task("Task 1", "Need to complete python assignment", "Completed")
manager.add_task("Task 2", "Go to grocery store", "Pending")
manager.add_task("Task 3", "Watch new Imax movie Lord of the Rings: Kings
Return", "Pending")
manager.add_task("Task 4", "prepare for the interview with Google", "Pending")
manager.add_task("Task 5", "Go to the gym", "Completed")

# Get all the tasks with status "Pending"
pending_tasks = manager.get_task_by_status("Pending")

# Print all the pending tasks
print("\nPending Tasks\n")
for pending_task in pending_tasks:
    print(f"{pending_task.title} - {pending_task.description} -
{pending_task.status}")

# Change the status of Task 2 to "Completed"
manager.change_task_status("Task 2", "Completed")
print(f"\nStatus of Task 2 : {[task.status for task in manager.queue if
task.title == 'Task 2'][0]}")


# Store all the tasks in a CSV file
manager.store_task_in_csv()
```

Explanation:

First I initialize TaskManager then add several tasks by calling add_task method. Next by calling get_task_by_status method, list down all the task which is in pending state by passing "Pending" as the argument to the method. Next change the status of Task 2 , "Pending" → "Completed" state by calling change_task_status method. Next print Task 2 status to verify the change. Finally, by calling store_task_in_csv method store the tasks list in tasks.csv file.

Output:

$ python exercise_03.py

Pending Tasks

Task 2 - Go to grocery store - Pending

Task 3 - Watch new Imax movie Lord of the Rings: Kings Return - Pending

Task 4 - prepare for the interview with Google - Pending

Status of Task 2: Completed

tasks.csv file:

A1 | Task 1

	A	B	C	D
1	Task 1	Need to complete python assingment	Completed	
2	Task 2	Go to grocery store	Completed	
3	Task 3	Watch new Imax movie Loard of the Rings: Kings Return	Pending	
4	Task 4	prepare for the interview with Google	Pending	
5	Task 5	Go to the gym	Completed	
6				
7				
8				
9				

Exercise 04: Shopping Cart

Code:

```
class Product:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

class ShoppingCart:
    def __init__(self):
        self.products = []

    def add_product(self, item):
        product = {}
        product["name"] = item.name
        product["price"] = item.price
        product["quantity"] = item.quantity

        self.products.append(product)

    def remove_product(self, name):
        for product in self.products:
            if product["name"] == name:
                self.products.remove(product)

    def calculate_total_price(self):
        total = 0
        for product in self.products:
            total += product["price"] * product["quantity"] # total =
product_price * product_quantity keep adding until all products are calculated
        return total

    def print_cart(self):
        for product in self.products:
            print(f"{product['name']} - {product['price']} -
{product['quantity']}")


# Create a few products
p1 = Product("MSI Laptop", 300000, 1)
p2 = Product("Pen drive", 1200, 3)
```

```python
p3 = Product("Mechanical Keyboard", 12500, 2)
p4 = Product("Windows 11 Pro", 15000, 1)

# Create a shopping cart
cart = ShoppingCart()

# Add products to the cart
cart.add_product(p1)
cart.add_product(p2)
cart.add_product(p3)
cart.add_product(p4)


# Remove Mechanical Keyboard from the cart
cart.remove_product("Mechanical Keyboard")

# Print cart
cart.print_cart()

# calculate total price
print("\nTotal Price : ", cart.calculate_total_price())
```

Explanation:

First, I created some products as objects. Then I initialize the ShoppingCart as cart object. Then using add_product method I added those created products. Each of the products will be stored as list of dictionaries. Then I remove "Mechanical Keyboard" from the cart using remove_product method. Then I print the cart items by calling print_cart method. Finally, by calling calculate_total_price method I calculate the total amount to pay.

Output:

$ python exercise_04.py

MSI Laptop - 300000 - 1

Pen drive - 1200 - 3

Windows 11 Pro - 15000 - 1

Total Price : LKR 318600

Exercise 05: Movie Database

Code:

```
class Movie:
    def __init__(self, title, genre, rating):
        self.title = title
        self.genre = genre
        self.rating = rating

class MovieDatabase:
    def __init__(self):
        self.movies = []
        self.unique_genres = set()

    def add_movie(self, movie):
        self.movies.append(movie)
        self.unique_genres.add(movie.genre) # add genre to the set of unique
genres

    def sort_movies_by_rating(self):
        self.movies.sort(key=lambda x: x.rating, reverse=True) # sort the movies
by rating in descending order
        return self.movies

    def search_movie_by_genre(self, genre):
        return [movie for movie in self.movies if movie.genre == genre] # create
a list of movies with the given genre

# create some movie objects
m1 = Movie("Harry Potter", "Fantasy", 8.1)
m2 = Movie("The Lord of the Rings", "Fantasy", 8.8)
m3 = Movie("Spider Man", "Action", 9.0)
m4 = Movie("Jonny English", "Comedy", 6.2)
m5 = Movie("The love", "Romance", 7.6)

# create a movie database object
movie_db = MovieDatabase()

# add the movies to the database
movie_db.add_movie(m1)
movie_db.add_movie(m2)
movie_db.add_movie(m3)
```

```python
movie_db.add_movie(m4)
movie_db.add_movie(m5)

# sort the movies by rating
sorted_movies = movie_db.sort_movies_by_rating()
print("Movies sorted by rating:")
for movie in sorted_movies:
    print(f"{movie.title}: {movie.rating}")

# search for movies with the genre "Fantasy"
fantasy_movies = movie_db.search_movie_by_genre("Fantasy")
print("\nFantasy movies:")
for movie in fantasy_movies:
    print(f"{movie.title}: {movie.rating}")

# print the unique genres
print("\nUnique genres:")
for genre in movie_db.unique_genres:
    print(genre)
```

Explanation:

As usual, first create 5 movie objects. Then initialize movie database and add those movies to the database using add_movie method. Next, sort movies by ratings in descending order using lambda function and inbuild sort function. Using search_movie_by_genre method I search movies of “Fantasy” genre and print the list one by one. Finally, print the Unique genres which is stored in a set data structure.

Output:

```
$ python exercise_05.py
Movies sorted by rating:
Spider Man: 9.0
The Lord of the Rings: 8.8
Harry Potter: 8.1
The love: 7.6
Jonny English: 6.2

Fantasy movies:
The Lord of the Rings: 8.8
Harry Potter: 8.1

Unique genres:
Fantasy
Action
Romance
Comedy
```

Exercise 06: Phone Book

Code:

```
import os

class Contact:
    def __init__(self, name, phone, email):
        self.name = name
        self.phone = phone
        self.email = email

class PhoneBook:
    def __init__(self):
        self.contacts = {}

    def add_contact(self, contact):
        try:
            if self.contacts[contact.name]:
                return f"{contact.name} already exists in the contacts"
        except:
            self.contacts[contact.name] = {"phone": contact.phone, "email":
contact.email}
            self.save_contact({"name": contact.name, "phone": contact.phone,
"email": contact.email})

    def del_contact(self, name):
        del self.contacts[name]

    def search_contact(self, name):
        if self.contacts and self.contacts[name]:
            return self.contacts[name]
        else:
            return f"{name} not found in the contacts"

    def save_contact(self, contact):
        with open("contacts.txt", "a") as file:
            file.write(f"{contact['name']}, {contact['phone']},
{contact['email']}\n")

    def load_contacts(self):
        with open("contacts.txt", "r") as file:
            for line in file:
                name, phone, email = line.strip().split(", ")
                if name not in self.contacts:
                    self.contacts[name] = {"phone": phone, "email": email}
```

```python
# Create some contacts
contact1 = Contact("Pawan", "1234567890", "pawan@gmail.com")
contact2 = Contact("Pinsara", "9577305765", "pinsara@gmail.com")
contact3 = Contact("Perera", "4561237890", "perera@gmail.com")
contact4 = Contact("Perera", "4561237890", "perera@gmail.com")

# Create a phone book object and add the contacts
phone_book = PhoneBook()

if os.path.exists("contacts.txt"):
    # Load the contacts from the file
    phone_book.load_contacts()
    print(phone_book.contacts)

phone_book.add_contact(contact1)
phone_book.add_contact(contact2)
phone_book.add_contact(contact3)
phone_book.add_contact(contact4)

# Search for a contact "Pawan"
print(phone_book.search_contact("Pawan"))

# Delete the contact "Pinsara"
phone_book.del_contact("Pinsara")

print(phone_book.contacts)
```

Explanation:

First create some contacts. Then initialize phonebook object. Using add_contact method add previously created contacts to the phone book instance. Before adding I am checking whether contacts.txt file exists. If it exists, load the contacts in it and store it in the dictionary. Then by calling search_contact method I search the contact by name. Using del_contact method we can delete any contact by just specifying the contact's name. Finally, I print the contacts to verify the contact successfully deleted.

Output:

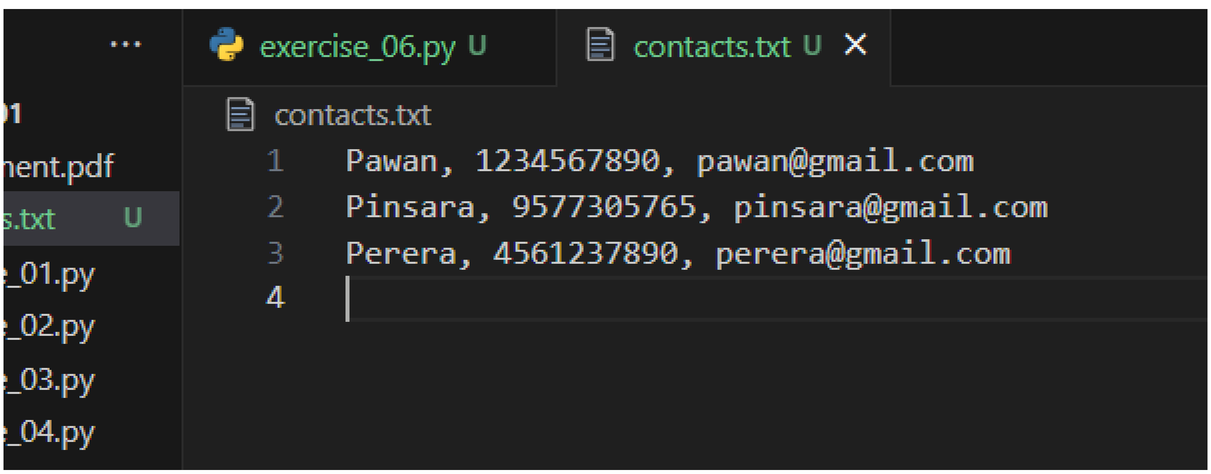
$ python exercise_06.py

{

'Pawan': {'phone': '1234567890', 'email': 'pawan@gmail.com'},

```
  'Pinsara': {'phone': '9577305765', 'email': 'pinsara@gmail.com'},
  'Perera': {'phone': '4561237890', 'email': 'perera@gmail.com'}
}
{
  'phone': '1234567890',
  'email': 'pawan@gmail.com'
}
{
  'Pawan': {'phone': '1234567890', 'email': 'pawan@gmail.com'},
  'Perera': {'phone': '4561237890', 'email': 'perera@gmail.com'}
}
```

contacts.txt:



Exercise 07: Stack Implementation

Code:

```
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        return self.stack.pop()

    def is_empty(self):
        return len(self.stack) == 0

def string_balance_check(string):
    """
    A string is balance if it has equal number of opening and closing
parenthesis. Thats what this function checks.
    """
    stack = Stack()

    for character in string:
        if character == "(":
            stack.push(character)
        elif character == ")":
            if stack.is_empty():
                return False
            stack.pop()

    return stack.is_empty()

string1 = "((()))"
string2 = "(()"
string3 = "())"
string4 = "()()"
string5 = "((()())())"

print(string_balance_check(string1))
print(string_balance_check(string2))
print(string_balance_check(string3))
print(string_balance_check(string4))
print(string_balance_check(string5))
```

Explanation:

In this code snippet I implement normal stack class which consist of push, pop and is_empty basic methods. Next use this created stack class to test the string with parenthesis is balanced. The logic is if the character is “(“ we push to the stack. And if the character is “(“ we pop value from the stacks. By this we can validate there is same number of opening and closing parenthesis are used.

Output:

$ python exercise_07.py

True

False

False

True

True

Exercise 08: Queue Implementation

Code:

```
class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item):
        """add item to the end of the queue"""
        self.queue.append(item)

    def dequeue(self):
        """remove item from the front of the queue"""
        if len(self.queue) == 0:
            return None
        return self.queue.pop(0)

    def is_empty(self):
        return len(self.queue) == 0 # this return boolean value


def ticketing_system(customers):
    queue = Queue() # create a queue object

    for customer in customers:
        queue.enqueue(customer) # add customer to the queue
        print(f"{customer} has been added to the queue.")

    print("\nServing customers in the following order:\n")

    while not queue.is_empty():
        customer = queue.dequeue()

        print(f"{customer} is being served.")


customers = ["Pawan", "Sahan", "Kavindu", "Pinsara", "Saman"]
ticketing_system(customers)
```

Explanation:

In ticketing_system function accepts customers list and add one by one to the queue using enqueue method. Then it serves the customer by First in First serve way using dequeue method. The function constantly checks whether the queue is empty using is_empty method. And serve all the customers in the queue in order.

Output:

```
$ python exercise_08.py
Pawan has been added to the queue.
Sahan has been added to the queue.
Kavindu has been added to the queue.
Pinsara has been added to the queue.
Saman has been added to the queue.

Serving customers in the following order:

Pawan is being served.
Sahan is being served.
Kavindu is being served.
Pinsara is being served.
Saman is being served.
```

Exercise 09: Recipe Book

Code:

```
import os
import json

class Recipe:
    def __init__(self, name, ingredients, instructions):
        self.name = name
        self.ingredients = ingredients
        self.instructions = instructions

class RecipeBook:
    def __init__(self):
        self.recipes_dict = {}

    def add_recipe(self, recipe):
        """
        First check if the file 'recipes.json' exists.
        If it does, open it and load the content to the variable
         self.recipes_dict. Then, update the dictionary with the new recipe.
        Finally, write the updated dictionary to the file 'recipes.json'.
        If the file does not exist, create it and write the recipe to it.
        The format of the dictionary should be as follows:
        {
            'recipe_name': {
                'ingredients': ['ingredient_1', 'ingredient_2', ...],
                'instructions': ['instruction_1', 'instruction_2', ...]
            }
        }
        """

        _recipe = {recipe.name: {'ingredients': recipe.ingredients,
'instructions': recipe.instructions}}

        try:
            if os.listdir('recipes.json'):
                with open('recipes.json', 'r') as f:
                    self.recipes_dict = json.load(f)
        except:
            pass

        self.recipes_dict.update(_recipe)

        with open('recipes.json', 'w') as f:
```

```python
            json.dump(self.recipes_dict, f)



    def search_recipe(self, name):
        """
        In this function, search for a recipe by name. If the recipe is found,
return the recipe object. If not, return None.
        """
        for recipe in self.recipes_dict:
            if recipe == name:
                return Recipe(recipe, self.recipes_dict[recipe]['ingredients'],
self.recipes_dict[recipe]['instructions'])

        return None

    def remove_recipe(self, name):
        del self.recipes_dict[name]

    def print_recipes(self):
        for recipe in self.recipes_dict:
            print(recipe)
            print("Ingredients: ", self.recipes_dict[recipe]['ingredients'])
            print("Instructions: ", self.recipes_dict[recipe]['instructions'])
            print("\n")

# create some recipes
recipe1 = Recipe("Tea", ["Tea bag", "Water"], ["Boil water", "Steep tea bag in
water", "Add sugar if desired"])
recipe2 = Recipe("Omlette", ["Eggs", "Salt", "Pepper", "Cheese", "Tomato",
"Onion"], ["Beat eggs", "Chop vegetables", "Add vegetables to eggs", "Cook on low
heat"])
recipe3 = Recipe("Dhal Curry", ["Lentils", "Water", "Turmeric", "Salt", "Pepper",
"Chili powder", "Curry leaves"], ["Boil lentils", "Add spices", "Cook until
thick"])

# create a recipe book
book = RecipeBook()

# add the recipes to the recipe book
book.add_recipe(recipe1)
book.add_recipe(recipe2)
book.add_recipe(recipe3)

# search for a recipe
```

```python
recipe = book.search_recipe("Omlette")
print("Recipe found: ", recipe.name)
print("Ingredients: ", recipe.ingredients)
print("Instructions: ", recipe.instructions)

# remove a recipe
book.remove_recipe("Omlette")
print("\nRecipe removed\n")

# print all recipes
book.print_recipes()
```

Explanation:

First, I create several recipes. Then after initializing RecipeBook object I added those recipes to the instance using add_recipe method. Then search and print the recipe using search_recipe method. Next, remove the recipe 'Omlette' from the dictionary using remove_recipe method. Finally, print all the existing recipes in the dictionary. Added recipes to the JSON file while adding the recipes to the dictionary.

Output:

```
$ python exercise_09.py
Recipe found:  Omlette
Ingredients:  ['Eggs', 'Salt', 'Pepper', 'Cheese', 'Tomato', 'Onion']
Instructions:  ['Beat eggs', 'Chop vegetables', 'Add vegetables to eggs', 'Cook on low heat']

Recipe removed

Tea
Ingredients:  ['Tea bag', 'Water']
Instructions:  ['Boil water', 'Steep tea bag in water', 'Add sugar if desired']


Dhal Curry
Ingredients:  ['Lentils', 'Water', 'Turmeric', 'Salt', 'Pepper', 'Chili powder', 'Curry leaves']
Instructions:  ['Boil lentils', 'Add spices', 'Cook until thick']
```

Recipes.json:

recipes.json

File Edit View

```json
{
    "Tea": {
        "ingredients": [
            "Tea bag",
            "Water"
        ],
        "instructions": [
            "Boil water",
            "Steep tea bag in water",
            "Add sugar if desired"
        ]
    },
    "Omlette": {
        "ingredients": [
            "Eggs",
            "Salt",
            "Pepper",
            "Cheese",
            "Tomato",
            "Onion"
        ],
        "instructions": [
            "Beat eggs",
            "Chop vegetables",
            "Add vegetables to eggs",
            "Cook on low heat"
        ]
    },
    "Dhal Curry": {
        "ingredients": [
            "Lentils",
            "Water",
            "Turmeric",
            "Salt",
            "Pepper",
            "Chili powder",
            "Curry leaves"
        ],
        "instructions": [
            "Boil lentils",
            "Add spices",
            "Cook until thick"
        ]
    }
}
```

Exercise 10:

Code:

```
import os

class WeatherData:
    def __init__(self):
        self.weekly_temperatures = ()

        try:
            if os.listdir("weather_data.csv"):
                with open("weather_data.csv", "r") as file:
                    for line in file:
                        self.weekly_temperatures += (float(line),)
        except:
            pass

    def add_temperature(self, temperature):
        self.weekly_temperatures += (temperature,)
        # store data in a file
        with open("weather_data.csv", "a") as file:
            file.write(str(temperature) + "\n")

    def get_average_temperature(self):
        return sum(self.weekly_temperatures) / len(self.weekly_temperatures)

    def get_max_temperature(self):
        max = float('-inf')

        for temperature in self.weekly_temperatures:
            if temperature > max:
                max = temperature

        return max

    def get_min_temperature(self):
        min = float('inf')

        for temperature in self.weekly_temperatures:
            if temperature < min:
                min = temperature

        return min

    def print_weekly_temperatures(self):
```

```
        DAYS = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"]
        print()
        for day, temperature in zip(DAYS, self.weekly_temperatures):
            print(day, ":", temperature)

# add temperatures to the file
weather_data = WeatherData()

weather_data.add_temperature(20.8)
weather_data.add_temperature(25.3)
weather_data.add_temperature(30.0)
weather_data.add_temperature(35.7)
weather_data.add_temperature(40.2)
weather_data.add_temperature(45.6)
weather_data.add_temperature(50.9)

print("\nAverage temperature:", weather_data.get_average_temperature())
print("\nMax temperature:", weather_data.get_max_temperature())
print("\nMin temperature:", weather_data.get_min_temperature())

weather_data.print_weekly_temperatures()
```

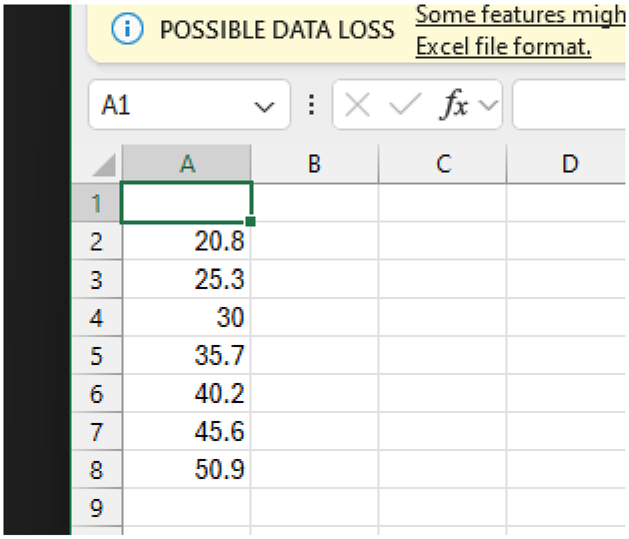
Explanation:

First initialize the WeatherData object. In the constructor program load the data from weather_data.csv if it is available. Otherwise each and every entry data is storing to the csv file. Next I added several temperature for one week. Then by calling get_average_temperature, get_max_temperature, get_min_temperature obtain average, min and max temperatures of the week respectively.

Output:

```
$ python exercise_10.py

  Average temperature: 35.5

  Max temperature: 50.9

  Min temperature: 20.8

  Monday : 20.8
  Tuesday : 25.3
  Wednesday : 30.0
  Thursday : 35.7
  Friday : 40.2
  Saturday : 45.6
  Sunday : 50.9
```

Weather_data.csv:



	A	B	C	D
1				
2	20.8			
3	25.3			
4	30			
5	35.7			
6	40.2			
7	45.6			
8	50.9			
9				